A

Efficient Implementation of Multigrid Solvers on Message-Passing Parallel Systems

(For presentation at SHPCC94)

Presenting author: John Lou

Notification to: John Lou

Address:

John Lou
Mail Stop: 169-315
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

lou@acadia.jpl.nasa.gov
Tel: (818) 354-4870
Fax: (818) 393-4802

Efficient Implementation of Multigrid Solvers on Message-Passing Parallel Systems

Extended Abstract

We discuss our implementation strategies for finite difference multigrid partial differential equation (PDE) solvers on message-passing systems. Our target parallel architecture is Intel parallel computers: the Delta and Paragon system. It is shown that natural grid decompositions for 2D **and** 3D problems mapped onto ring, 2D and 3D logical meshes of physical processors can achieve scalable performances for multigrid V-cycle and full V-cycle, algorithms. It is also shown how grid partitions, local data structures, logical processor networks for fine and coarse grids and code modules are chosen to maximize the efficiency and robustness of the code. Performances of our code (written in C) on Delta and Paragon machines are (will be) presented and analyzed.

1. Multigrid Algorithm and Its Parallel Implementation

Multigrid solvers are efficient iterative solvers for many PDE problems due to its ability to reduce numerical errors of all frequencies effectively by performing relaxations on an hierarchy of fine to coarse grids. A typical coarse-grid-correction scheme. consists of three major components: relaxation on a given grid, restriction of residuals to a coarser grid and interpolation of errors back to a finer grid. In a geometrical (as opposed to algebraic) multigrid scheme, interpolation and restriction operators can be defined in a simple and intuitive way and therefore writing a sequential code for a classical multigrid (e.g. V-cycle) scheme is relatively simple. 'J'here have been some the.orc.tical analysis and implementations of parallel multigrid schemes [1] [2]. One difficulty that arises in the parallel implementation of a complete V-cycle. scheme is the idle processor issue. Not only may the idle processors degrade the efficiency of processor usage, but on a distributed-memory system, message-passing structure may also need to change for processing on a coarse grid. When the number of fine grid points contained in each processor is not too small compared to the dimension of the processor network used, the efficiency of processor usage should be reasonably good. In our implementation of V-cycle type schemes, the original (fine) grid is partitioned and distributed to an initial processor network. Local coarse. subgrids in a processor are derived from local finer subgrids recursively. Processing on a coarse grid when idle processors appear is done by using a corresponding coarse, processor network,

Multigrid solver is often used as a component in some numerical schemes like multilevel or adaptive mesh algorithms. It is there.fore important to develop a multigrid solver that is highly modularized, robust and extensible so that either it can be easily incorporated into some application code or it can be extended into an application code without much difficulty. These are the principles we followed in our code development. As a result, the code we developed can be easily adapted to solving linear, nonlinear and time-dependent PDEs.

2. Our Implementation Strategy

A) Computational Grid Partitions and Logical Processor Networks

For a 2D problem, a 1D or 2D grid partition can be used. *in* a 1D strip partition of a M x N computational grid to P processors, we configure the processors into a ring and let each processor contain M/P consecutive rows of grid points if M is divisible by P; otherwise first M

$mod(P)$ processors get one more row. This grid partition and processor network are illustrated in figure 1. For a 3D problem, 2D or 3D grid partition can be used. Figure 2 shows a partition of a 3D grid mapped to a 2D logical processor network, Figure 3 shows a partition of 3D grid assigned to a 3D logical processor network.

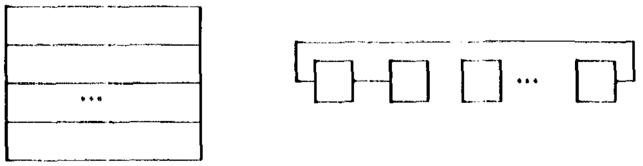

Figure 1: 2D grid partition mapped to a ring processor network

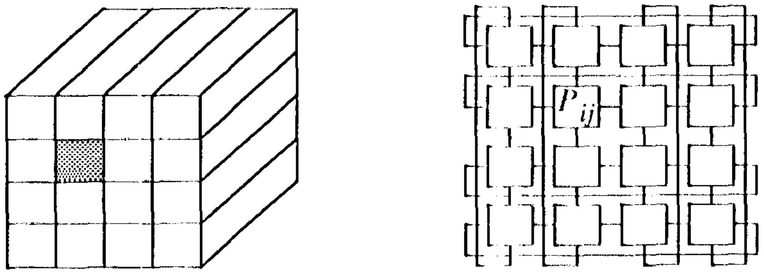

Figure 2: 3D grid partition mapped to a 2D torus mesh processor network

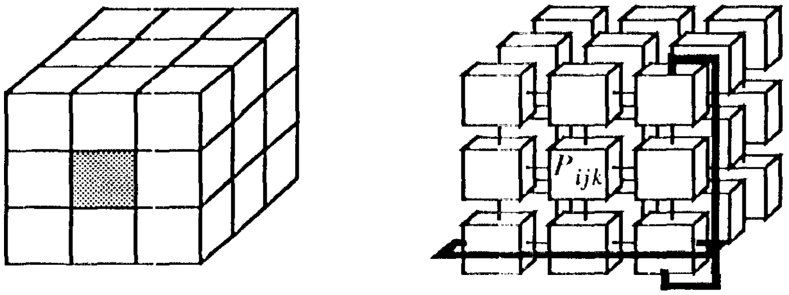

Figure 3: 3D grid partition mapped to a 3D torus mesh processor network; only two wrap-around connections are shown in the network.

The wrap-around logical connections in 1, 2 and 3D processor networks are useful in constructing coarse processor networks for coarse grid processing; but they are not used in multi-grid iterations. For a d-dimensional computational grid and P processors, a natural processor network to use seems to be a d-dimensional mesh. But as the computational grid gets coarser and coarser, the number of grid points contained in each processor may decrease to a point that message-passing cost dominates the local computation cost. Using a (d-1)-dimensional processor mesh for a d-dimensional computational grid, the minimum number of computational grid points each processor can have is larger because some processors will become idle earlier in the V-cycle iterations than using a d-dimensional processor mesh. Another practical issue is the number of message-passing and buffer-copying operations required to exchange grid partition boundary data using a (d-1)-dimensional grid partition is smaller, However, **the processor** utilization at later stages of a V-cycle scheme is larger for the d-dimensional grid partition. Therefore it seems not obvious which partition strategy would give a better performance. (d-1)-dimensional grid partition can also be advantageous when line relaxation is used in one dimension for some asymmetric PDE because the data locality in one dimension makes the line relaxation a local operation.

B) Local Data Structures

Each processor maintains a structure to hold a list of fine to coarse grids. An array of pointers to a data type "ndim_grid" is declared to hold **solutions** defined on different local grids, where ndim = 1, 2 or 3. The same data structure. is used to store right-hand side vectors on different grids. Figure 4 shows this type of data structure which allows efficient access to grid values on different grids. Integer arrays are used to store indices in each dimension for each local subgrid. These indices are needed for multigrid processing and for message-passing to get partition boundary information. Information related to grids is collected in a single structure **Grid.** Other information needed at various stages (e.g. neighbor processor IDs, processor status etc.) is collected into another data structure. **Misc.** Pointers to these structures are passed, when needed, to various functions so that interfaces between functions are very simple and the number of global variables is minimized.

To enhance code robustness and efficiency of memory usage, only pointers are defined in data structures for grid vectors anti indices. Storages required for different grids are calculated and allocated at run-time in a preprocessing module.

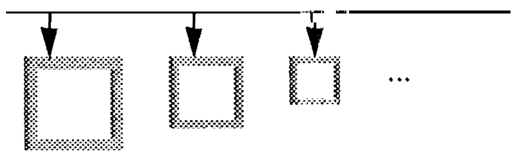

Figure 4: Data structure for solution and right-hand side vectors on different grids. Shaded bounding areas are used to store partition boundary values, and interior areas store local grid values.

C) Interprocessor Communications

In doing message-passing, each processor only communicates with its nearest neighbors. To implement complete V-cycle/ full V-cycle. iterations on one. of the processor networks discussed above, one needs to deal with coarse grid processing in which some processors may become idle at certain stage. This implies as gridcoarsening reaches a certain stage, the original processor network can not be used anymore. since for active processors at this stage, their nearest neighbors may have changed. We chose to construct an hierarchy of coarse logical processor networks to be used cm coarse grids when idle processors appear. All logical processor networks arc constructed in a preprocessing module and this processing module is called only once. Since V-cycle or full V-cycle iteration routines are usually called many times, especially in solving a time-dependent PDE, the cost of executing the preprocessing module should be relatively 10W. Message-passing for exchanging partition boundary values is needed for three operations in a multigrid cycle: relaxation, residual calculation for restriction and interpolation for correction. For relaxation, message-passing with a nearest interior neighbor processor is symmetric; but for restriction and correction, the needed message-Jxtssing is generally not symmetric.

1)) Code Modules

Our multigrid code consists of two main modules: a pre.processing or set-up module and a multigrid cycle module. The preprocessing module takes care of user input, program initialization, construction of logical processor networks, local memory allocations and assignment of boundary conditions. The multigrid cycle module performs V-cycle or full V-cycle iterat ions through recursive function calls. An array of local flags indicates whether a processor is active or idle at each stage of processing, that is used by both set-up and multigrid cycle modules to control the processor's participation in various operations.

3. Performances

The performances of our code on Delta machine (performances on Paragon will be reported later) for 3D grids using two grid partitions for solving Poisson equation with 1)irichlet boundary condition arc displayed in table 1 and 2. Execution time shown is for 1 V-cycle (MV) or 1 full V-cycle (FMV) where red-black Gauss-Seidel scheme is used for smoothing. It is seen that in both cases the performance scales reasonably well. The set-up time is for running the preprocessing module whose cost is more related to the local grid din iensions and structure in each processor than to the size of the global processor net woi k.

Table 1: 3D Grid on 3D Logical Mesh Processor Networks

Grid size and network	64^3 grid on 1 processor	1283grid on $2 \times 2 \times 2$ network	256^3 grid on 4X4X4 net work	5123grid on $8 \times 8 \times 8$ net work
set-llp	1.8 sec	37.6 sec	38.2 sec	40.0 sec
1 MV	9.1 sec	11.7 sec	12.5 sec	13.3 sec
1 FMV	11.8 sec	15.9 sec	16.7 sec	17,6 sec

Table 2 31) Grid on 2 1) Logical Mesh Processor Networks

Grid size and network	64^3 grid on 1 processor	1283grid on 2.X4 network	256^3 grid cm 8×8 net work	5123grid on 16X32 net work
set-up	1.8 sec	9.5 sec	11.9 sec	6.8 sec
1 MV	9.1 sec	10.9 sec	11.7 sec	13.() sec
1 FMV	11.8 Scc	15.3 sec	16.2 sec	17.2 sec

[1] Chan, F. C. and Tuminaro, R. S., "A Survey of Parallel Multigrid Algorithms", in Parallel Computations and Their Impact cm Mechanics, A. K. Noor ed., Vol: AM I)86, 1986

[2] McCormick, S. F., "Multilevel Adaptive Methods for Partial Differential Equations", SIAM Frontiers in Applied Mathematics, Vol. 6, Philadelphia, 1989